



ELSEVIER

Theoretical Computer Science 264 (2001) 25–51

Theoretical
Computer Science

www.elsevier.com/locate/tcs

Logical string rewriting

Vincenzo Manca

Università degli studi di Pisa, Dipartimento di Informatica, Corso Italia, 40-56125 Pisa, Italy

Accepted April 2000

Abstract

A logical analysis of string manipulation systems is presented that provides a unification of many formalisms and suggests a framework for the investigation of complex discrete dynamics. Namely, several characterizations of computational universality are given in terms of logical representability within models and theories; moreover, *combinatorial* schemata, as a formal counterpart of DNA basic recombinant mechanisms, are logically expressed. In this way a general definition of *derivation* systems is given to which many classical systems can be easily reduced, and where some regulation mechanisms can be naturally represented. As a further consequence, systematic methods are provided for translating derivation systems into *monoidal* theories. Finally, new possibilities of this logical approach are outlined in the formalization of molecule manipulation systems inspired by chemical and biochemical processes. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: String rewriting; Formal systems; Formal languages; Grammars; Logical representability; DNA computing; Molecular computing

1. Introduction

String rewriting is the basis of formal language theory, where symbolic systems seem to amount, according to reasonable taxonomic and enumerative criteria, to several thousands of different types. Without being exhaustive, we point out some fundamental *phyla* of grammatical models and related formalisms.

Logistic and deductive calculi [39] were the early historical formal systems. Strongly related to them, there were Curry, Lorenzen and Church's formal calculi [8, 24, 38], which tried to individuate the inductive structure of hypothetic-deductive and computational processes. Post's systems [32] were the first attempt to put the matter in terms of string manipulation; in fact they provided the symbolic form of axiomatic systems.

E-mail address: mancav@di.unipi.it (V. Manca).

Chomsky grammars [5] introduced a sort of simplified form of Post systems, based on substring pattern matching and substring replacement. Originally motivated by studies in structural linguistics, they could be considered as a cornerstone in the conceptual path, from which the classic formal language theory stems with its fundamental concepts and results [35, 36].

Since the 1960s other important grammatical formalisms were introduced in formal language theory by Marcus [27], Lindenmayer [22], Head [15], and others [6, 11, 30, 31, 35] with a lot of variants and related formalisms, and with strong motivations from linguistics and biology. If we consider Chomsky's work, and the original context of Kleene's theorem on regular sets [19], we realize the centrality of linguistic and biological contents also in classic formal language theory [28].

All these systems exploit different, possibly cooperative, parallel and distributed strategies, but they have two essential peculiarities: (i) they manipulate strings; (ii) each of these systems has a derivation relation by means of which a language is derived, and this language constitutes the most important evaluation parameter of the system.

We claim that it would be important to develop an *intensional* analysis of *string rewriting* in the more general framework of *derivation*. The attribute 'intensional' is used in opposition to the classical *extensional* analysis; this is essentially focused on the languages that systems generate and on the localization of these languages in Chomsky's hierarchy, or in other hierarchies. An intensional study of string manipulation systems is naturally connected with a logical formalization of them, and could suggest guidelines in the formal analysis of more general forms of discrete dynamical systems which arise in physical, chemical, biochemical, or biological contexts.

In these systems strings, or better (symbolic) molecules, are naturally *classified* in different types (possibly in many undistinguishable copies), are *aggregated* into bigger units (macromolecules, or other finite structures), and are *localized* in different *membranes* (possibly at many levels), inside which some effects can be confined.

Formal languages, with their recent developments related to biomathematics [1, 2, 13, 15, 18, 23, 31] and for their centrality in representation and computation [28], seem to be the natural setting where new mathematical models can be found which are adequate to study complex, discrete dynamics [12, 14].

In this paper, by extending an approach initiated in [25, 26], we present a unifying logical perspective in the analysis of string manipulation systems, aimed to investigate the more general concept of a *molecule* manipulation system.

In Section 2 we show the possibility of representing many syntactical concepts within the first-order model *SEQ* based on (numerical) strings.

In Section 3 we develop the notion of logical representability within first-order theories and present a logical characterization of type-0 languages.

In Section 4 we introduce *monoidal representability*, a particular case of *axiomatic representability* that seems to be a logical counterpart of the notion of grammar. Monoidal representability is strictly related to Smullyan's *elementary formal systems* [38], which result to be monoidal theories in Horn logic, and recently were used for

modeling concepts of DNA computing (splicing systems and multiple splicing systems [20]).

In Section 5 we present a general definition of (string) *derivation system*. This definition is based on the notion of *combinatorial schema* over an alphabet, a concept strictly related to Post rules, that seems to be the most simple way to describe abstractly many operations typical of DNA molecules (grammars based on combinatorial operations are studied in [9, 10] with motivations arising from genome evolution).

The general definition of derivation systems is the basis of Section 6 where we develop monoidal representations of important classes of string rewriting systems, and give some *adequacy theorems* for the given representations. In the final section we analyze the notion of *metabolic system*, inspired by chemical and biochemical processes, and outline how the same logical approach of previous sections can be applied also to the logical representation of this concept.

2. Logical representability

Assume the seven logical symbols (*if then, not, and, or, iff, for all, there exists*):

$$\rightarrow, \neg, \wedge, \vee, \leftrightarrow, \forall, \exists$$

with the standard syntactical and semantical first-order logical notions (equality predicate $=$ is assumed in its usual usage) that can be found in introductory treatises or basic chapters of textbooks in mathematical logic [3, 37].

We recall that \models is the predicative formalization of proof consequence \Rightarrow , according to the specific sense developed by mathematical logic. More precisely this symbol has two different, though related, meanings. We write

$$\mathcal{M} \models \varphi$$

to mean *validity w.r.t. models*, inasmuch as it expresses that the formula φ holds in the model \mathcal{M} , assuming that all the symbols of φ can be interpreted in \mathcal{M} (φ is a Σ -formula, where Σ is the signature of the model \mathcal{M}). A natural extension of this meaning is

$$\mathcal{M} \models \Phi$$

where Φ is a set of formulae, which expresses that $\mathcal{M} \models \varphi$ holds for any formula φ of Φ .

The second meaning of \models is the first-order *logical validity*. In this sense

$$\Phi \models \varphi$$

means that for all first-order models \mathcal{M} (more exactly, for all Σ -models \mathcal{M} , if Φ is a Σ -theory)

$$\mathcal{M} \models \Phi \Rightarrow \mathcal{M} \models \varphi.$$

Given a formula $\varphi(x)$ with a free variable x and an individual term t , we indicate by $\varphi(t)$ the formula obtained from $\varphi(x)$ by replacing in it all the occurrences of x with the individual term t and assuming that no variable of t is bound in $\varphi(t)$.

Definition 2.1. Let Σ be a signature and \mathcal{M} be a Σ -model. A subset \mathcal{S} of the domain of \mathcal{M} is represented in \mathcal{M} by the formula $\varphi(x)$ when

$$a \in \mathcal{S} \Leftrightarrow \mathcal{M} \models \varphi(a).$$

Note that $\varphi(a)$ is generally a formula in the signature which extends Σ with the element a as a new constant interpreted by itself.

In the same manner we could define the representability of properties and relations within a given model.

Let AR be the *standard arithmetical model*

$$AR = (\omega, +, \cdot, 0, 1)$$

where ω is the set of natural numbers, $+$ and \cdot denote the plus and times operations on natural numbers, respectively, and 0 and 1 denote numbers zero and one, respectively (the formula $\exists z(x + z = y)$ represents in AR the usual ordering relation $x \leq y$ between natural numbers). As usual, we may use, ambiguously, the same symbols to indicate either the symbols of a signature Σ , or the corresponding functions, relations and individuals of a Σ -model.

Notation 2.1. $x \in \mathcal{S} \equiv_{\mathcal{M}} \varphi$ indicates that φ represents \mathcal{S} within \mathcal{M} . When the model \mathcal{M} is clearly understood, the subscript \mathcal{M} may be omitted.

Example 2.1. The set P of prime numbers is represented in AR , by the following formula in the first-order language of signature $\{+, \cdot, 0, 1\}$:

$$x \in P \equiv_{AR} \forall yz(\neg x = 0 \wedge \neg x = 1 \wedge (x = y \cdot z \rightarrow x = y \vee x = z)).$$

A model which is very useful in representing syntactical notions is the following:

$$SEQ = (\omega^*, --, ||, 0, \lambda)$$

where ω^* is the set of finite strings of natural numbers, $--(\alpha, \beta)$ abbreviated by $\alpha\beta$ is the concatenation of strings α and β , $|\alpha|$ is the length of string α , and $0, \lambda$ are the constants for zero and the empty string. In this model numbers can be conceived as symbols of an infinite alphabet ω , therefore, any language can be embedded in its domain (we may identify $0, 1, 2, \dots$ with letters a, b, c, \dots).

We write: $\alpha(i) = n$ to say that symbol (number) n occurs in the string α at position i , $\alpha \leq \beta$ to say that α is a substring of β ; the constant 1 stands for $|0|$.

Example 2.2.

$$y \in \omega \equiv (|y| = |0|)$$

$$x(i) = y \equiv \exists zw(x = zyw \wedge y \in \omega \wedge |zy| = i)$$

$$x \preceq y \equiv \exists zw(y = xzw)$$

$$x + y = z \equiv \exists uv(x = |u| \wedge y = |v| \wedge z = |uv|).$$

Example 2.3. The language $\{a^n b^n \mid n \in \omega\}$ is SEQ-representable:

$$\alpha \in \{a^n b^n \mid n \in \omega\} \equiv \exists xy(\alpha = xy \wedge |x| = |y| \wedge$$

$$\forall uv((u \preceq x \wedge |u| = 1 \rightarrow u = a) \wedge (v \preceq y \wedge |v| = 1 \rightarrow v = b)).$$

How can we represent the product between natural numbers in terms of relationships among strings? We can get a solution to this question by representing the natural process of calculating a product by means of iterated sums. In fact if $m \cdot k = n$, there exists a sequence of natural numbers $m, m \cdot m, m \cdot m \cdot m, \dots$, of length k that ends with n .

Example 2.4.

$$m \cdot k = n \equiv$$

$$\exists w(|w| = k \wedge w(1) = m \wedge$$

$$\forall x(1 \leq x < k \rightarrow w(x+1) = w(x) + m) \wedge w(k) = n).$$

The idea of the previous example has a wide application. Moreover, notice that the simple structure of the formula representing multiplication is mainly due to the possibility of *SEQ*-representing computations directly (thus avoiding all kinds of syntax encoding, e.g. Gödel's β -function, cf. [37]).

The same idea can be used for representing in the model *SEQ*, the k th prime number, the power m^k , and the factorial $n = m!$.

Let us assume the classical notions of formal language theory, for further details see [35, 36]. We only recall briefly some basic definitions in order to fix the notation.

We can consider languages over alphabet A as particular subsets of the free monoid ω^* . This embedding allows us to represent syntactic properties in the model *SEQ*.

Let us consider a Chomsky grammar $G = (T, N, P, S)$, where T is the finite alphabet of terminals, N is the finite set of nonterminals, P is the finite set of productions $\alpha \rightarrow \beta$, with $\alpha \in V^+$, $\beta \in V^*$ (V^* being the free monoid generated by $V = T \cup N$, and $V^+ = V^* - \{\lambda\}$) and, finally, $S \in N$ is the start symbol. From S and replacing iteratively in a string the left side of a production with the right one, a grammar G generates a type-0 language $L(G)$ constituted by all $\alpha \in T^*$ generated from S with a finite number of replacements. Type-0 languages coincide with the recursively enumerable subsets of

T^* , i.e. the subsets generated by some process that is effective w.r.t. some universal computation formalism.

Consider the following syntactical relation ($\exists!$ means *there exists exactly one*):

- $sub(\alpha, i, j, \beta)$ means that the string β is the substring of α starting at position $i + 1$ and ending at position j ($i \leq j$, if $i = j$ then $\beta = \lambda$), shortly

$$\alpha[i, j] = \beta$$

- $occur(\alpha, a, n)$ means that in the string α the symbol a occurs exactly n times, shortly

$$|\alpha|_a = n$$

- $hom_f(\alpha, \beta)$ means

$$f(\alpha) = \beta$$

where f is a string homomorphism.

- $perm(\alpha, \beta)$ means that α is a permutation of β .

Lemma 2.1. *Syntactical relations: sub , $occur$, $perm$, and hom_f are SEQ-representable.*

Proof.

- sub

$$\begin{aligned} \alpha[i, j] = \beta &\equiv (i = j \rightarrow \beta = \lambda) \wedge \\ &(i < j \rightarrow \exists \gamma \delta (\alpha = \gamma \beta \delta \wedge |\gamma| = i \wedge |\gamma \beta| = j)) \end{aligned}$$

- $occur$

$$\begin{aligned} |\alpha|_a = n &\equiv \exists v (|v| = n \wedge \forall i (1 \leq i \leq n \rightarrow \alpha(v(i)) = a)) \\ &\wedge \forall i (\alpha(i) = a \rightarrow \exists! j (v(j) = i)) \end{aligned}$$

- $perm$

$$perm(\alpha, \beta) \equiv |\alpha| = |\beta| \wedge \forall x (x \leq \alpha \wedge |x| = 1 \rightarrow |\alpha|_x = |\beta|_x)$$

- hom_f

$$\begin{aligned} f(\alpha) = \beta &\equiv (\alpha = \lambda \rightarrow \beta = \lambda) \wedge (\alpha \neq \lambda \rightarrow \\ &\exists v (|v| = |\alpha| + 1) \wedge \forall i (1 \leq i \leq |\alpha| \rightarrow \beta[v(i), v(i + 1)] = f(\alpha(i))))). \quad \square \end{aligned}$$

The following theorem shows the logical expressivity of the model SEQ.

Theorem 2.2. *Any type-0 language is SEQ-representable.*

Proof. Let L be a language generated by a type-0 grammar G . A string α belongs to L if there exist two strings β and v , the first one constituted by the concatenation

of all strings of a derivation of α , the second one (whose length is the length of the derivation + 1) being a vector that allows us to extract from β , for every pair of values of consecutive indices, the derivation steps of α . In order to characterize all strings of L and only them, it is sufficient to express that, for any derivation of them, any step is obtained from the previous one by applying some production of G . This is easily representable by a simple logical condition expressed by concatenation and existential quantification. \square

3. Standard syntax

Is any *SEQ*-representable language a type-0 language? If not, which class of formulae represents type-0 languages? In order to answer these questions, we will extend the notion of first-order representability.

Let us recall some basic notions about first-order theories. A set Φ of Σ -formulae is *recursively enumerable* (r.e.) or *semidecidable* if we have an algorithm that can effectively generate all the formulae of Φ . It is *decidable* or *recursive* iff we have an algorithm for deciding when a given Σ -formula belongs to Φ or not, or equivalently, iff we can effectively generate all the formulae of Φ and all the Σ -formulae that do not belong to Φ . A Σ -theory Φ is a set of formulae closed w.r.t. the first-order logical consequence, that is, for every Σ -formula: $\Phi \models \varphi \Rightarrow \varphi \in \Phi$. If AX is a decidable (recursive) set, then $\Phi = \{\varphi | AX \models \varphi\}$ is an *axiomatic* theory of *axioms* AX (if AX is finite, Φ is finitely axiomatizable).

Given a Σ -model \mathcal{M} , we indicate by $TH(\mathcal{M})$ the Σ -theory of all Σ -formulae that are true in the model \mathcal{M} . *Atomic* formulae are constituted by relation symbols applied to terms, or by equations. The *diagram* $DIAG(\mathcal{M})$ of a model \mathcal{M} is constituted by the atomic formulae, or negated atomic formulae which belong to $TH(\mathcal{M})$. A theory Φ is *axiomatizable* iff $\Phi = \{\varphi | AX \models \varphi\}$ for some r.e. set of axioms AX . Given the effective nature of first-order logical consequence (for the existence of complete deductive calculi), the set of theorems of an axiomatizable theory is recursively enumerable.

A Σ -theory Φ is *sound* iff it is not the case that $\Phi \models \varphi$ and $\Phi \models \neg\varphi$ for some Σ -formula φ (it is *complete* iff always one of two condition holds).

We can extend the notion of logical representability of sets (and relations) by the following definition. Let T_Σ be the set of terms without variables on the signature Σ .

Definition 3.1. Let Φ be a theory over the signature Σ . A subset \mathcal{T} of T_Σ is represented in Φ , by the formula $\varphi(x)$ when

$$t \in \mathcal{T} \Leftrightarrow \Phi \models \varphi(t).$$

When a set is representable within a theory Φ by a formula φ that belongs to a class Γ of formulae, we say that it is Φ -representable by Γ .

An interesting case of representability within a theory is *axiomatic* representability:

Definition 3.2. A subset \mathcal{T} of T_Σ is Φ -representable by a finite set of axioms AX if for some formula φ in the signature of $\Phi \cup AX$:

$$t \in \mathcal{T} \Leftrightarrow \Phi \cup AX \models \varphi(t).$$

Of course, representability within a theory and axiomatic representability can naturally be extended to any relation.

Let us indicate by SS the theory of *Standard Syntax*, constituted by the diagram of the model SEQ . This theory is strongly related to Raphael–Robinson’s arithmetical theory RR (R^- in [37]). We use symbols \leq and \leqslant (meaning the usual order on natural numbers and the substring inclusion, respectively) as abbreviations, of their representations, in terms of $+$, $--$. Consider the following formulae:

$$\forall x(x \leq t \rightarrow \varphi)$$

$$\forall x(x \leqslant t \rightarrow \varphi).$$

In these formulae universal quantifications are \leq -bounded, or \leqslant -bounded (\leq and \leqslant are the bounding predicates, while t is the bounding term); we say also that a usual quantification is an *unbounded* quantification. A formula φ over the signature of AR is a Σ_1 -formula iff any universal quantification that occurs in it is a \leq -bounded quantification. Analogously, a formula φ over the signature of SEQ is a Σ_1 -formula iff any universal quantification that occurs in it is a \leq -bounded or \leqslant -bounded quantification.

The most important fact about Σ_1 -formulae of AR is a theorem strictly connected to Kleene’s Normal Form Theorem in Computability theory (cf. Theorem I.11.7 and related topics in [37]): *A set L of natural numbers is recursively enumerable iff it is AR -representable by some Σ_1 -formula.* This theorem is an easy consequence of the following result:

Theorem 3.1 (Σ_1 -completeness of RR). *Let φ be a Σ_1 -formula over the signature of AR , with a free variable and let $n \in \omega$, then:*

$$AR \models \varphi(n) \Leftrightarrow RR \models \varphi(n).$$

Proof. By induction, see Theorem III.6.13 in [37]. \square

If we use for the model SEQ , the same arguments used in the Σ_1 -completeness of RR , we get these results:

Theorem 3.2 (Σ_1 -completeness of SS). *Let φ be a Σ_1 -formula over the signature of SEQ , with a free variable and let $\alpha \in \omega^*$, then:*

$$SEQ \models \varphi(\alpha) \Leftrightarrow SS \models \varphi(\alpha).$$

Proof. By induction, analogously to the proof of the previous theorem. \square

Theorem 3.3. *A language L is type-0 iff it is SEQ-representable by a Σ_1 -formula.*

Proof. If L is SEQ-representable by some Σ_1 -formula, then it is also SS-representable because SS is Σ_1 -complete. Therefore, L is type-0, because SS is an axiomatic theory and its theorems form a recursively enumerable set. Also, if L is type-0, then, by Theorem 2.2 it is SEQ-representable; moreover by the proof of that theorem, it follows that such representability can be expressed by a Σ_1 -formula. \square

Theorem 3.4. *A language L is type-0 iff it is SS-representable by some Σ_1 -formula.*

Proof. By the Σ_1 -completeness of SS and the previous theorem. \square

This logical characterizations of type-0 languages are very close to an analogous representation theorem formulated in terms of *rudimentary predicates* (cf. [36, Chapter, III Theorem 12.5; 38]). A rudimentary predicate is essentially determined by a Σ_1 -formula where no symbol $||$ occurs. The representation of a language via rudimentary predicates is equivalent to its representability by Σ_1 -formulae, with no occurrence of $||$ within the following model:

$$(\omega^*, \lambda, -, -, \varpi)$$

where ϖ denotes a predicate that holds on atomic strings (i.e. single numbers). This equivalence is a straightforward consequence of the next theorem.

Theorem 3.5. $|\alpha| = |\beta|$ is representable in the model $(\omega^*, \lambda, -, -, \varpi)$ by a Σ_1 -formula (with no occurrence of $||$).

Proof. Consider, for example, two strings with the same length: $\alpha = abc$ and $\beta = cbb$; associate to them the strings α' and β' :

$$\alpha' = a\#ab\#\#abc\#\#\#$$

$$\beta' = c\#cb\#\#cbb\#\#\#.$$

These are obtained, step by step, adding one symbol at time of α and β , respectively, in the order they appear, and by separating consecutive substrings of α and β with strings of $\#$ of increasing length. Therefore, we can express that α and β have the same length by saying that α' and β' ends with the same substring of consecutive $\#$.

Assume that α and β are nonempty and nonatomic. Let us say that a string is *monic* iff it is constituted only by occurrences of the symbol $\#$, and that a monic string γ is a *full substring* of a string δ when it occurs in δ as a substring that is not a proper substring of another monic string. Under these assumptions $|\alpha| = |\beta|$ iff there are two strings α', β' that satisfy the following requirements:

- both α' and β' begin with one symbol followed by $\#$, and end with a monic substring;
- a monic γ can occur only once in α' as its full substring;
- a monic γ is a substring of α' iff it is a substring of β' ;

- iff $\gamma\#$ is a monic full substring of α' , then also γ is a full substring of α' which precedes $\gamma\#$ (from the left);
- given a monic γ , let $\gamma\#\#$, $\gamma\#$, and γ be full substrings of α' and β' , and let α_1, α_2 and β_1, β_2 be the pairs of strings such $\gamma\alpha_1\gamma\#\alpha_2\gamma\#\# \leq \alpha'$ and $\gamma\beta_1\gamma\#\beta_2\gamma\#\# \leq \beta'$. In this case: α_2 is obtained from α_1 by adding to it only one (rightmost) symbol; analogously, β_2 is obtained from β_1 by adding to it only one (rightmost) symbol;
- α and β are the two strings that in α' and β' are between the two rightmost monic full substrings;
- the two ending monic strings of α' and β' are equal.

It is easy to verify that we can put together all these conditions in a Σ_1 -formula. \square

4. Monoidal representability

A further specialization of axiomatic representability is that of *monoidal representability* defined as follows. Let A be a finite set of *symbols*, called an alphabet, and let Σ be a signature that includes the symbols of A as constants plus another constant λ (for the empty string) and a binary function symbol (for concatenation, which we indicate by juxtaposition). A theory Φ of signature Σ is said to be *monoidal* over the alphabet A , if the terms of Φ without variables are obtained by concatenation from A and λ , and moreover Φ contains the usual axioms *MON* of *monoid* (x, y, z are variables)

$$\forall x y z (x(yz) = (xy)z)$$

$$\forall x (x\lambda = x \wedge \lambda x = x)$$

(associativity and null element). This means that terms of Φ without variables coincide with A^* , that is, with the free monoid generated by the alphabet A .

Definition 4.1. Let Φ be a monoidal Σ -theory over the alphabet A , AX the axioms of Φ different from *MON* (called *proper axioms* of Φ), and φ a Σ -formula with only one free variable. A language over the alphabet A is monoidally representable by the pair (AX, φ) if:

$$\alpha \in L \Leftrightarrow \Phi \models \varphi(\alpha).$$

The following are examples of monoidal representability.

Example 4.1. $\{a^n b^n \mid n \in \omega\} \subseteq \{a, b\}^*$ is monoidally representable by (AX, L) , where AX are the following axioms (in the signature $\{-- , \lambda, a, b, L\}$).

- $L(\lambda)$
- $\forall x (L(x) \rightarrow L(axb))$

In fact: $\alpha \in \{a^n b^n \mid n \in \omega\} \Leftrightarrow MON \cup AX \models L(\alpha)$.

Example 4.2. $\{a^n b^n c^n \mid n \in \omega\} \subseteq \{a, b, c\}^*$ is analogously representable by considering the following axioms AX :

- $L(\lambda)$
- $L(abc)$
- $\forall x y (L(xby) \rightarrow L(axbbbyc))$

In fact: $\alpha \in \{a^n b^n c^n \mid n \in \omega\} \Leftrightarrow MON \cup AX \models L(\alpha)$.

Example 4.3. The set of prime numbers is monoidally representable by (Φ, Prime) :

$$\alpha \in \{a^p \mid p \text{ is a prime number}\} \Leftrightarrow \Phi \models \text{Prime}(\alpha)$$

with the following proper axioms: (in the signature $\{-, \lambda, a, \text{Factor}, \text{Less}, \text{Prime}\}$):

- $\forall x \text{Factor}(x, x)$
- $\forall x y (\text{Factor}(x, y) \rightarrow \text{Factor}(x, xy))$
- $\forall x \text{Less}(x, ax)$
- $\forall x y (\text{Less}(x, y) \wedge \text{Less}(y, z) \rightarrow \text{Less}(x, z))$
- $\forall x (\forall y (\text{Less}(y, x) \wedge \neg a = y \rightarrow \neg \text{Factor}(y, x)) \rightarrow \text{Prime}(x))$.

The following theorem establishes the computational universality of monoidal representability.

Theorem 4.1. *Any type-0 language is monoidally representable.*

Proof. Let G be the grammar of a type-0 language L with terminal symbols a_1, \dots, a_n and start symbol S ($\alpha \rightarrow \beta$ stands for any production of G). Consider the following axioms AX in a signature which includes the symbols of G (as constants) plus the symbols for the concatenation and two unary predicates D, T :

$$\begin{aligned} & D(S) \\ & T(a_1) \wedge T(a_2) \dots \wedge T(a_n) \\ & \forall x y (D(x\alpha y) \rightarrow D(x\beta y)) \\ & \forall x y (T(x) \wedge T(y) \rightarrow T(xy)). \end{aligned}$$

In this manner we have: $\alpha \in L \Leftrightarrow MON \cup AX \models D(\alpha) \wedge T(\alpha)$. \square

Vice versa, if a language is monoidally representable, then it is a type-0 language. This is a consequence of recursive enumerability of the theorems of any axiomatic theory.

5. Derivation and rewriting

The notion of monoidal theory allows us to develop a very general and simple analysis of basic string manipulation mechanisms.

Definition 5.1. Let Σ be the signature of a monoidal theory over an alphabet A . A combinatorial (k, m, n) -schema r over A consists of k distinct variables u_1, \dots, u_k called

parameters of r ; m Σ -terms t_1, \dots, t_m called premises of r ; and n Σ -terms s_1, \dots, s_n called conclusions of r . Each parameter has to occur in some premise or conclusion of r . The schema r is completely described by

$$r(u_1, \dots, u_k): \frac{t_1, \dots, t_m}{s_1, \dots, s_n}$$

where $r(u_1, \dots, u_k)$ makes explicit the parameters, and the premises/conclusions expression represents the combinatorial mechanism of r . If parameters are instantiated by the strings $\alpha_1, \dots, \alpha_k$ of A^* , we say that the schema determines a (combinatorial) rule of type (m, n) and we indicate it by

$$r(\alpha_1, \dots, \alpha_k): \frac{p_1, \dots, p_m}{q_1, \dots, q_n}$$

where p_1, \dots, p_m , the instances of t_1, \dots, t_m , are called premises or the rule, and q_1, \dots, q_n , the instances of s_1, \dots, s_n , are called conclusions or the rule. When also the other variables of the schema (the internal variables) are instantiated, and strings $\beta_1, \dots, \beta_m, \gamma_1, \dots, \gamma_n \in A^*$ are the instances of premises and conclusions respectively, we write

$$r(\alpha_1, \dots, \alpha_k): \frac{\beta_1, \dots, \beta_m}{\gamma_1, \dots, \gamma_n}$$

and we say that strings $\gamma_1, \dots, \gamma_n$ are obtained by applying rule $r(\alpha_1, \dots, \alpha_k)$ to strings β_1, \dots, β_m .

A combinatorial rule of type $(1, 1)$ is said to be a (combinatorial) rewriting rule.

The following is a list of combinatorial schemata which some important string manipulation formalisms are based on.

$$\begin{aligned} \text{replace}(u, w): \frac{xuy}{xwy}, \quad \text{insert}(w, u, v): \frac{xuy}{xuwy}, \quad \text{insert2}(w, u, v): \frac{xwy}{xuwy}, \\ \text{delete}(z): \frac{xzy}{xy}, \quad \text{duplicate}(u): \frac{xuy}{xuuy}, \quad \text{transpose}(u, v): \frac{xuy}{xvuy}, \\ \text{translocate}(u, w, v): \frac{xuzwtvy}{xutwzvy}, \quad \text{rotate}(w): \frac{xwy}{ywx}, \quad \text{extract}(u, v): \frac{xuwy}{w}, \\ \text{cut}(u, v): \frac{xuy}{xu, vy}, \quad \text{paste}(u, v): \frac{xu, vy}{xuy}, \quad \text{splice}(u, v, u', v'): \frac{xuy, zu'v'w}{xuv'w, zu'vy}. \end{aligned}$$

Many of the previous combinatorial schemata have a natural biochemical interpretations as operations involved in DNA recombination, and in genome evolution [9, 10, 15, 16].

Given a set R of rules over an alphabet A and a set B of strings of A^* , the set

$$A(B, R)$$

of *derivations* of *axioms* B and rules R consists of finite sequences δ such that $\delta(1) \in B$ and, if n is the length of δ , for every $1 < i \leq n$ one of the following conditions has to hold:

- $\delta(i) \in B$
- $\delta(i)$ is obtained by the application of some rule of R to some strings of δ which occur at positions preceding i .

If the last element $\delta(n)$ of a derivation δ is the string α we write

$$\delta \vdash \alpha$$

and we write

$$\vdash \alpha$$

if there exists some derivation δ such that $\delta \vdash \alpha$.

Two derivations of $\Delta(B, R)$ are said to be *equivalent* if they end with the same string. Assume that R is a set of $(m, 1)$ -rules. A derivation $\delta \in \Delta(B, R)$ of length n is said to be *linear* when, for any $1 \leq i < n$, if the element $\delta(i+1)$ is obtained by applying a rule of R , then $\delta(i)$ is among the premises of this application.

Lemma 5.1 (Linearity lemma). *If R is a set of $(m, 1)$ -rules. Given a derivation $\delta \in \Delta(B, R)$ there is always a linear derivation equivalent to it.*

Proof. By induction on the length of derivations. For derivations where no rules are applied the linearity is trivial. Assume that for derivations of length smaller than n we have linear derivations equivalent to them. If δ is a derivation of length $n+1$ we have two possibilities: (i) the string $\delta(n)$ is among the premises by means of which $\delta(n+1)$ is obtained; (ii) it is not. In the first case δ is linear, in the other case if we remove $\delta(n)$ from δ we get a derivation of length n that is equivalent to δ , therefore, by the inductive hypothesis it has a linear equivalent derivation. \square

The notion of combinatorial rule and of derivation allow us to give the main definition of this section.

Definition 5.2. A derivation system \mathcal{D} is given by

$$\mathcal{D} = (A, T, B, R, D)$$

where A is a finite alphabet and T a subset of A , of terminal symbols; B is a set of initial strings over the alphabet A ; R is a finite set of combinatorial rules over the alphabet A , and D is a subset of $\Delta(A, R)$ of well-formed derivations. The language $L(\mathcal{D})$ derived by \mathcal{D} is given by

$$L(\mathcal{D}) = \{\alpha \in T^* \mid \delta \vdash \alpha, \delta \in D\}.$$

A derivation system $\mathcal{D} = (A, T, B, R, D)$ is said to be:

- *pure* if $T = A$;
- *extended* if $T \subset A$;
- *regulated* if $D \subset \Delta(B, R)$;
- *unregulated* if $D = \Delta(B, R)$;
- *r-homogeneous* if all the rules of R are instances of the combinatorial schema r over A ;
- (m, n) -ruled if all its rules have at most m premises and at most n conclusions;
- *deterministic* if for any $\delta \in \Delta(B, R)$ there exists at most one derivation that can be obtained by extending δ with an application of some rule of R ;
- *local* if it is $(1, 1)$ -ruled and $\vdash x \Rightarrow \vdash y$ implies $\vdash uxv \Rightarrow \vdash uyv$.
- *finitary* if B is a finite set;
- *infinite* if $L(\mathcal{D})$ is infinite (*finite* otherwise).

According to the previous definitions, pure and unregulated derivation systems can be identified by omitting T and D , respectively. Moreover, in a finitary derivation system, the set B of initial strings can be expressed by rules without premises (*input* rules), that is, a finitary derivation system \mathcal{D} can be identified by a quadruple $\mathcal{D} = (A, T, R, D)$. Finally, if we adopt the usual convention of indicating non-terminal symbols by capital letters and terminals by lower case letters, then a finitary derivation system can be identified by a pair (R, D) of rules and derivations (only by R if it is also unregulated).

Lemma 5.2 (Rewriting representation lemma). *For any finitary derivation system there exists an extended $(1, 1)$ -ruled derivation system that derives the same language.*

Proof. Introduce two extra symbols $\#, *$ and the following rules for any symbol a of the original system (x, y are variables)

$$\frac{x \# y}{x \# * \# y} \quad \frac{x * y}{xa * y} \quad \frac{x * y}{xy}$$

then, encode any rule

$$\frac{p_1, \dots, p_m}{q_1, \dots, q_n}$$

of the system, with variables x_1, \dots, x_k , into a rule with 1 premise and 1 conclusion

$$\frac{x \# p_1 \# p_2 \dots \# p_m \# x_1 \# \dots \# x_k \# y}{x \# p_1 \# p_2 \dots \# p_m \# q_1 \# q_2 \dots \# q_n \# y}.$$

Finally, encode all the axioms $\alpha_1, \dots, \alpha_n$ into a unique string $\# \alpha_1 \# \dots \# \alpha_n \#$, and add to the system the rules *translocate*($\#, \#, \#$) and *extract*($\#, \#$). The rule *translocate* allows the changing of the positions of the components in a string (put between two consecutive $\#$), so that the encoding of a rule could be applied; the rule *extract* allows the derivation, in the new system, of strings derivable in the initial system. The three rules added initially, and the encoding of the original rules allow us to distinguish the

variables instantiated with the strings of the original system, from the variables of the new system (confusing them would give an unsound translation). \square

A *rewriting system* is a pair $\mathcal{R} = (A, \rightarrow)$ where A is a (finite) alphabet and \rightarrow is a binary relation over the free monoid generated by A . A set R of combinatorial rules of type $(1, 1)$ over A determines a (finitary) rewriting system over A . This system becomes automatically a derivation system when terminals, axioms, and derivations are fixed. On the other hand, any $(1, 1)$ -ruled derivation system \mathcal{D} determines a rewriting system on the alphabet A . To this end, it is sufficient to put for every $\alpha \in A^*$, $\alpha \rightarrow \beta \Leftrightarrow \vdash \alpha \Rightarrow \vdash \beta$.

Chomsky grammars are unregulated, finitary, derivation systems that are homogeneous w.r.t the combinatorial schema. A finite state automaton $M = (T, Q, q_0, F, \sigma)$ with alphabet T , states Q , initial schema state q_0 , final states F , and transition relation σ is an unregulated derivation system $(T \cup Q, T, \{q_0\}T^*, R)$ with the following rules R ; the first of which for any q, a, q' such that the transition $\sigma(q, a, q')$ holds, the second one for any $q_f \in F$:

$$\frac{xqay}{xaq'y}, \quad \frac{xq_f}{x}.$$

In fact, the language derived by this system coincides with the language recognized by M .

Derivation systems which describe other automata, transducers, Marcus' contextual grammars, Head's splicing systems, and many other formalisms can be trivially obtained whenever combinatorial schemata are identified which they are based on; moreover, in many cases, this identification is immediate. Lindermayer's systems and regulated grammars will be considered in the next section.

Let C be a finite set of combinatorial rules, and let \mathcal{C} be the class of unregulated derivation systems which have rules in C . We say that C is *computationally universal* if $RE = \{L(\mathcal{D}) \mid \mathcal{D} \in \mathcal{C}\}$. It is easy to see that combinatorial rules with only one conclusion are computationally universal. A deeper universality result is the following, from which Kuroda's normal form theorem follows easily (see [31] for the relevance of this normal form in DNA computing).

Consider an alphabet A , a *2-replacement* over A is one of the following rules where x, y are variables, $\alpha, \beta \in A^*$, $|\alpha| \leq 2$, and $|\beta| \leq 2$

$$\frac{x\alpha y}{x\beta y}, \quad \frac{x\alpha}{x\beta}, \quad \frac{\alpha x}{\beta x}.$$

Theorem 5.3. *The set of 2-replacement rules is computationally universal.*

Proof. Given a Turing machine M , an output of M is the string on the tape of M when it halts, after deleting the blank symbols that are at beginning on the left side (before the first non-blank symbol) and at the end on the right side (after the last non-blank symbol). Let $L(M)$ be the language of the strings that we can get as output of M in correspondence to all possible input strings. We know that if TM is the class of Turing machines, then $RE = \{L(M) \mid M \in TM\}$. Therefore, our proof is concluded

if we show that, given a Turing machine M we can define an unregulated derivation system \mathcal{D}_M whose rules are 2-replacements and which derives the language $L(M)$. Let A be the input symbols of M , Q its states, q_0 its initial state, a_0 its blank symbol, q_f its final state, and I the instructions of M expressed as strings of 4 symbols: *state*, *read symbol*, *new state*, *written symbol* or *right/left move* (r/l). Put

$$\mathcal{D}_M = (A \cup Q \cup \{\bar{q} \mid q \in Q\} \cup \{q_g\}, A, \{S\}, R).$$

The rules R are the following where x, y are variables, $a, b, c \in A$; $c \neq a_0$; $q \in Q$; and rules r_5, r_6, r_7 are relative to any overwriting rule $qapb$, to any right move $qapr$, and to any left move $qapl$, respectively

$$\begin{aligned} r_1: & \frac{Sx}{Sax}, & r_2: & \frac{Sx}{q_0x}, & r_3: & \frac{x}{a_0x}, & r_4: & \frac{x}{xa_0}, & r_5: & \frac{xqay}{xpby}, \\ r_6: & \frac{xqay}{xapy}, & r_7: & \frac{xqay}{x\bar{p}ay}, & r_8: & \frac{x\bar{a}qy}{xqay}, & r_9: & \frac{x\bar{a}q_fy}{xq_fay}, \\ r_{10}: & \frac{q_fa_0x}{q_fx}, & r_{11}: & \frac{q_fcx}{cq_gx}, & r_{12}: & \frac{xq_gay}{xaq_gy}, & r_{13}: & \frac{xa_0q_g}{xq_g}, & r_{14}: & \frac{xcq_g}{xc}. \end{aligned}$$

Rules r_1, r_2 generate the input on the tape; rules r_3, r_4 enlarge the tape with blanks; rules r_5, r_6 simulate an overwriting and a right move of M ; and rules r_7, r_8 simulate a left move of M . The remaining rules delete the external blanks and produce the outputs of M . \square

6. Monoidal theories for derivation

The notion of an unregulated derivation system is essentially equivalent to the notion of a Post system inasmuch as a combinatorial rule is essentially a rule in the sense of Post. However, our formulation has several advantages, with respect to Post's classical definition, that we will illustrate in the sequel. The first advantage is expressed by the following theorem. For the sake of brevity, in the axioms of theories that we will present, free variables are intended to be implicitly universally quantified.

Theorem 6.1 (Logical translation of unregulated derivation systems). *For every unregulated derivation system \mathcal{D} there exists a monoidal theory $TH_{\mathcal{D}}$ and a formula φ in its signature such that $(TH_{\mathcal{D}}, \varphi)$ monoidally represents $L(\mathcal{D})$.*

Proof. For every rule of the system

$$\frac{p_1, p_2, \dots, p_m}{q_1, q_2, \dots, q_n}$$

consider the formula

$$Der(p_1) \wedge Der(p_2) \wedge \dots \wedge Der(p_m) \rightarrow Der(q_1) \wedge Der(q_2) \wedge \dots \wedge Der(q_n)$$

and for every axiom α of \mathcal{D} consider the formula $Der(\alpha)$; moreover, add the following formulae (x, y variables):

- $Term(x) \wedge Term(y) \rightarrow Term(xy)$
- $Term(x) \wedge Der(x) \rightarrow Lang(x)$.

The set of formulae so obtained constitute $TH_{\mathcal{D}}$, in fact we have

$$\alpha \in L(\mathcal{D}) \Leftrightarrow TH_{\mathcal{D}} \models Lang(\alpha)$$

this means that the pair $(TH_{\mathcal{D}}, Lang)$ represents $L(\mathcal{D})$. \square

The result of this theorem can be improved by defining a systematic translation of unregulated derivation systems. In fact we have:

Theorem 6.2 (Systematic logical translation). *Let C be a finite set of combinatorial schemata, and let \mathcal{C} be the class of unregulated derivation systems over an alphabet A which have some instances of C as rules. There are (i) axioms $TH_{\mathcal{C}}$ describing the derivation mechanism of the class \mathcal{C} , (ii) axioms $AX_{\mathcal{D}}$ which refer to a specific system $\mathcal{D} \in \mathcal{C}$, constituted by atomic formulae, and (iii) a formula ϕ such that for every $\mathcal{D} \in \mathcal{C}$, the pair $(TH_{\mathcal{C}} \cup AX_{\mathcal{D}}, \phi)$ monoidally represents $L(\mathcal{D})$.*

Proof. The theory $TH_{\mathcal{C}} \cup AX_{\mathcal{D}}$ is a monoidal theory over A , where the axioms $AX_{\mathcal{D}}$ are: (i) predicates that express which instances of schemata in C are rules of \mathcal{D} , (for example $Replace(\alpha, \beta)$ for every production $\alpha \rightarrow \beta$ if \mathcal{D} is a Chomsky grammar); (ii) the formulae $Term(a)$ for every terminal symbol of \mathcal{D} ; (iii) the formulae $Der(\alpha)$ for every axiom α of \mathcal{D} . The axioms of $TH_{\mathcal{C}}$ are

- $Rule(x_1, \dots, x_k) \wedge Der(t_1) \wedge \dots \wedge Der(t_m) \rightarrow Der(s_1) \wedge \dots \wedge Der(s_n)$
for every combinatorial schema

$$rule(x_1, \dots, x_k) : \frac{t_1, \dots, t_m}{s_1, \dots, s_n}$$

- $Term(x) \wedge Term(y) \rightarrow Term(xy)$
- $Term(x) \wedge Der(x) \rightarrow Lang(x)$.

From the given axioms it follows that $(TH_{\mathcal{C}} \cup AX_{\mathcal{D}}, Lang)$ represents $L(\mathcal{D})$. \square

The following monoidal theory CG gives a logical translation of Chomsky grammars (x, y, u, v variables):

1. $Replace(u, v) \wedge Der(xuy) \rightarrow Der(xvy)$
2. $Term(x) \wedge Term(y) \rightarrow Term(xy)$
3. $Term(x) \wedge Der(x) \rightarrow Lang(x)$

Given a Chomsky grammar G , $AX(G)$ are the following axioms (the alphabet A of G is the alphabet of the monoidal theory):

1. $Der(S)$ if S is the start symbol of G
2. $Term(a)$ for every terminal symbol of G
3. $Replace(\alpha, \beta)$ for every production $\alpha \rightarrow \beta$ of G .

The pair $(CG \cup AX(G), Lang)$ is a monoidal representation of $L(G)$.

Analogous theories can be obtained for automata, transducers, Marcus' grammars, and Head's systems, automatically, from their representation as derivation systems.

6.1. Monoidal representation of parallel rewriting

L-systems, with their main variants [33–35], are based on parallel rewriting mechanisms and express developmental aspects typical of growing processes of biological systems. The proper axioms of a monoidal theory *OL* for interactionless L-systems are:

1. $Par(x, y) \wedge Par(u, v) \rightarrow Par(xu, yv)$
2. $Gen(x) \wedge Par(x, y) \rightarrow Gen(y)$.

A particular L-system S over an alphabet A is identified by the axioms $AX(S) = \{Par(a, \alpha) \mid a \in A\}$ and by a formula $Gen(\alpha)$ telling the initial word of the system.

The proper axioms of a monoidal theory *EOL* for *extended* interactionless L-systems (where a distinction is made between terminal and non terminal symbols) are obtained in the usual manner, by adding to *OL* the following axioms:

1. $Term(x) \wedge Term(y) \rightarrow Term(xy)$
2. $Gen(x) \wedge Term(x) \rightarrow Lang(x)$.

The class of interactionless *tabled* L-systems is obtained by considering a rewriting which is the union of many parallel rewriting relations $\{Par_i \mid i = 1, \dots, k\}$, indexed by a finite set of indexes (*tables*). The proper axioms of a monoidal theory *ETOL* for these systems are obtained by replacing the first axiom of *OL* by the following axioms

- $Par_i(x, y) \wedge Par_i(u, v) \rightarrow Par_i(xu, yv)$
- $Par_i(x, y) \rightarrow Par(x, y)$.

The adequacy of these monoidal theories, with respect to the corresponding L-systems, is a simple consequence of the given axioms. However, the basic mechanism of Lindermayer systems can be expressed only by combinatorial rules. In fact we have the following theorem.

Theorem 6.3. *Let S be an L-system and $L(S)$ the language generated by S . There exists an unregulated, extended, derivation system \mathcal{D}_S that derives $L(S)$.*

Proof. Given the following parallel rewriting

$$a_1 \rightarrow \beta_1, \dots, a_n \rightarrow \beta_n$$

consider the following rules ($1 \leq i \leq n$; x, y variables):

$$\frac{x\#a_i y}{x\beta_i\#y}, \quad \frac{x\#}{x, \#x}.$$

The derivation system \mathcal{D}_S has the alphabet of S plus symbol $\#$, the same terminal symbols as S , the axioms $\{x\# \mid x \text{ axiom of } S\}$, and all the rules considered above. \square

6.2. Monoidal representation of regulated rewriting

Our method of logical representation can be extended to regulated derivation systems. Let us explain this point with some examples. Matrix grammars [11] are the oldest

example of regulated grammatical systems. In these grammars rules are productions as in Chomsky grammars, but they are organized in rows; a string is rewritten by entering it in a row and by applying to it all the productions of the row in their order (if a rule cannot be applied, the process aborts). If m_1, m_2, \dots, m_k are the rows of a matrix grammar, we can express that r is the index of the production $u \rightarrow v$ in the row m by a predicate $Row(m, r, u, v)$. The order of rules can be expressed by a predicate $Initial(m, r)$, telling that r is the initial rule of the row m , by a predicate $Next(m, r, q)$ telling that q is the rule following r in the row m , and by a predicate $Final(m, r)$ telling that r is the final rule of the row m .

The monoidal theory MT of matrix grammars is given by the axioms (x, y, u, v, m, r, q variables):

1. $Der(x) \wedge Initial(m, r) \rightarrow Localder(m, r, x)$
2. $Row(m, r, u, v) \wedge Localder(m, r, xuy) \wedge Next(m, r, q) \rightarrow Localder(m, q, xvy)$
3. $Localder(m, r, xuy) \wedge Final(m, r) \wedge Row(m, r, u, v) \rightarrow Der(xvy)$
4. $Term(x) \wedge Term(y) \rightarrow Term(xy)$
5. $Term(x) \wedge Der(x) \rightarrow Lang(x)$.

Let $AX(G)$ be the axioms describing a particular matrix grammar G (the rules of every row, the order of these rules, the start symbol and the terminal symbols); the following theorem establishes the adequacy of the theory MT with respect to matrix grammars, and provides a systematic translation method for the class of matrix grammars.

Theorem 6.4. $\alpha \in L(G) \Leftrightarrow MT \cup AX(G) \models Lang(\alpha)$

Proof. The first axiom above *enters* the string in order to be processed by the first rule of a row; the second axiom passes the control to the next rule of a row; the third axiom tells when all the rules of a row have been applied. The other axioms, as in the theory CG of Chomsky grammars, define the derivation, the terminal strings, and the language generated by G . \square

Regulation is a powerful tool for obtaining complex behaviours. However regulated systems can be very often transformed into unregulated ones if the initial combinatorial rules are transformed into rules with a more complex combinatorial mechanism. This phenomenon is illustrated by the following theorem.

Theorem 6.5. *Let G be a matrix grammar and $L(G)$ the language generated by G . There exists an unregulated, extended, derivation system \mathcal{D}_G that derives $L(G)$.*

Proof. For any row of G

$$r_1 : \alpha_1 \rightarrow \beta_1, \dots, r_n : \alpha_n \rightarrow \beta_n$$

consider the following rules ($1 \leq i < n$; x, y variables):

$$\frac{x}{r_1 x}, \quad \frac{r_i x \alpha_i y}{r_{i+1} x \beta_i y}, \quad \frac{r_n x \alpha_n y}{x \beta_n y}.$$

The derivation system \mathcal{D}_G has the alphabet of G , extended with symbols for the indexes of rules, the same terminal symbols and axioms as G , and all rules considered above, for any row of G . \square

In some cases, regulation has a more complex pattern that needs all the expressivity of monoidal theories. For example, in the case of (totally) *ordered* grammars productions are ordered:

$$r_1 : \alpha_1 \rightarrow \beta_1, \dots, r_n : \alpha_n \rightarrow \beta_n$$

and production r_i can be applied only if no production r_j for $j < i$ can be applied. We could express this strategy, that is common to Markov's normal algorithms (with some additional termination conditions) [36], by the following rules ($1 \leq i < n$; x, y variables, r_1, r_2, \dots, r_n nonterminals):

$$\frac{x}{r_1 x}, \quad \frac{r_i x \alpha_i y}{x \beta_i y}, \quad \neg \exists uv(x = u \alpha_i v) \Rightarrow \frac{r_i x}{r_{i+1} x}.$$

Here, in the last rule, we need a regulating predicate in order to express adequately the rewriting mechanism. This means that we are using monoidal representability in a sense which is stronger with respect to that used for unregulated systems. In other words, the previous example shows a Post system where rules are regulated by logical conditions. Of course, this regulation does not increase the computational power, because Post systems are computationally universal, but gives them more expressive power in the description of complex derivation strategies. However, even if the meaning of the condition of the last rule is clear, we need to be more precise in order to specify where its logical condition has to be evaluated. A natural solution is the following. Consider the theory Φ constituted by the following axiom (\preceq is a binary predicate for substring inclusion):

$$\forall x y u v (x \preceq x \wedge x \preceq y \rightarrow u x v \preceq u y v).$$

Now, what the rule intends to express can be rigorously represented if we replace the condition $\neg \exists uv(x = u \alpha_i v)$ by:

$$\varphi \equiv \neg \exists uv(u \preceq x \wedge v \preceq x \wedge x = u \alpha_i v)$$

and we say that the rule is regulated by φ within the theory Φ , that is, it can be applied when the formula φ is derivable in the theory $MON \cup \Phi$.

This example suggests us the general definition of Φ -regulated derivation system, as a finitary system $\mathcal{D} = (A, T, R, \Phi)$ where to every rule $r \in R$ is associated a regulation formula φ whose free variables occur in the premise of r , and Φ are the axioms added to the monoid axioms in order to specify the theory which regulation formulae refer to.

In this notion of regulation the logical theory is a sort of environment within which rules are defined. This makes the derivation strategy distributed among them; in other words, good derivations are obtained by applying the rules that fulfill some logical conditions.

In the recent theory of grammar systems [6, 29] and regulated rewriting [11] the idea of many (possibly different) local derivation relations integrated according to specific strategies, has been extensively applied with different patterns for developing grammatical approaches where simple components are integrated and monolithic complex systems are reduced, by means of cooperation and distribution, in terms of simpler parts. Monoidal representability provides natural tools for describing such complex systems. In fact, each component can be formalized by some axioms, and other axioms can be used to describe the mechanisms for integrating the different parts.

7. Logical metabolism

The language generated by a derivation system is the formal counterpart of the history of a development, where all the generations are put together. On the other hand, a rewriting system models a process of state transformation, where a string models a state and a rewriting relation expresses the passage to another state. The process of derivation is conservative because what has been derived at some step remains alive in the following steps. The process of rewriting in itself is transformative, because a rewritten string replaces the old one. In several physical, chemical, biochemical, and biological systems, phenomena present a behaviour that combine aspects common to derivation and rewriting. We call these *metabolic* systems. They are dynamical systems where, at any step, what is transformed is a finite set of strings, that is a language, called *metabolite*. Any metabolic transformation is based on a structural recombination of some constituents of a metabolite (some elements are destroyed and others are constructed according to a consuming/producing pattern). If an environment is also assumed to exist which adds new things (reagents, energy, food), then the size of the system can grow, and natural generative or developmental processes can be associated with it.

More specifically, consider different molecules, for example: HHO, COO, ATP, nucleotides, which can be represented by strings of a suitable alphabet (in this context HHO is the chemical type of a water molecule). Some of them are combined into bigger units called (macro) molecules or are grouped into solutions separated by membranes where they are continuously floating in a casual way. In a metabolic system there are essentially four types of possible reactions: (i) new units are created or modified by combining or recombining smaller units; (ii) some smaller units are obtained by destroying a bigger one; (iii) some elements transmigrate from the membrane where they are located to another one; (iv) some elements enter into the system from an external environment, and/or other elements go outside the system to the external environment. At any moment the system is completely specified by how many molecules of each type are present, how they are grouped into membranes, and which chemical types these molecules possess. Certain transformations can hold if some *catalysts* or *enzymes* are present, and more generally if also suitable conditions are verified.

This notion of metabolism is inspired by biochemical metabolism (cf. [21]), but of course it is completely idealized: several peculiar features of biochemical transformations are only abstractly considered or not directly accounted for in this model, e.g. the energetic aspects of transformations.

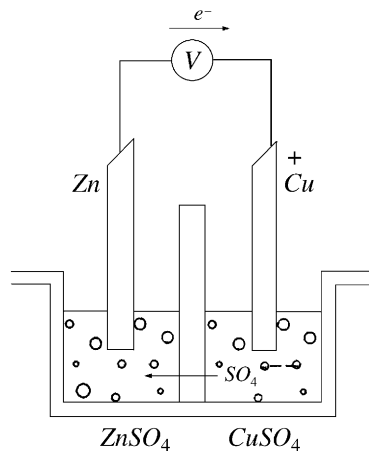
Interesting examples of (symbolic) metabolic systems, taken from chemical, and biochemical contexts, are in [4, 26]. In the string rewriting systems studied by Turing [40], quantitative parameters are also considered in order to show morphogenetic phenomena.

The notion of metabolic system is very general; in fact, any discrete dynamical system can be easily represented with a metabolic system for a suitable set of metabolites, and for a suitable metabolic relation.

A grammar can be easily considered as a metabolic system. In fact, if at any step i of a generative process, we keep all generated strings in a language L_i , then a rewriting step where a string β is derived from a string $\alpha \in L_i$ can be read as a generation of the language $L_{i+1} = L_i \cup \{\beta\}$ from the language L_i .

Consider a mechanical system of n bodies which are moving in the space. The state of the system is specified at any time by $6n$ numbers (three values for the position of each body and three values for its speed in the three directions). If any value is represented (with some approximation) by a suitable string, then the global state of the system is completely represented by a finite language.

The next example intends to give a detailed description of a chemical phenomenon by metabolic steps.



Example 7.1 (Daniell cell). This example is a description of the electrochemical process that happens in a particular galvanic cell, known as Daniell cell. It is constituted by a negative electrode of zinc (Zn), and a positive electrode of copper (Cu) connected by a conductor wire. The electrodes are put, respectively, in two solutions: A of zinc sulphate (ZnSO_4), and B of copper sulphate (CuSO_4). These solutions are separated by some device allowing migration of the ions only from B to A (but not in the

reverse side). The process is regulated by the electronegativity difference between the two electrodes, and by the concentrations of solutions (Nerst's equation). We do not enter in the physico-chemical details; the quantitative aspects of the phenomenon are given by the value, determined by the function f which will appear in the metabolic rules that follow (f can be approximated, in a discrete way, and within some range, by suitable tables). In a descriptive way, we can say that a small percentage of the zinc metal dissolves in solution A in the ionic form Zn^{2+} and, for each dissolved atom Zn, two electrons are liberated in the electrode Zn ($\text{Zn} \rightarrow \text{Zn}^{2+} + 2\text{e}^-$). The electrons accumulated in the negative electrode Zn flow, through the wire, to the positive electrode Cu, where the copper ions in solution B are attracted; so they accept the electrons received by the electrode Cu, and pass in the metal form ($\text{Cu}^{2+} + 2\text{e}^- \rightarrow \text{Cu}$). At this point, in solution B the SO_4^{2-} ions exceed the ions Cu^{2+} , while in the solution A the ions Zn^{2+} exceed the ions SO_4^{2-} ; therefore, a flow of SO_4^{2-} ions goes from B to A, in order to restore the electrical equilibrium of the two solutions, and the process can start again. The usual initial configuration of this process are two rods, of Zn and Cu, and the same concentration of the two sulphates in the solutions A and B. The process goes on until the metal zinc is not completely dissolved or the copper ions Cu^{2+} in solution B are not completely transformed into copper metal atoms.

We can formalize this process by means of metabolic formulae. In this case it is appropriate to use a *quantitative* representation of multisets where qX , denotes a quantity q of indistinct elements of type X . We do not claim to present a physico-chemical rigorous description of the process, but rather we want to focus the attention on a kind of metabolism with an intrinsic periodic nature, driven by a cycle of non-equilibrium conditions. In fact, the phenomenon is due to the cooperation of several agents: each of them alters continuously the equilibrium conditions and activates another agent; after some steps, the system reaches again the initial non-equilibrium condition, and so the cycle is repeated. The logical sequence of the four rules corresponds to the order in which they are written; A.SO_4^{2-} , B.SO_4^{2-} indicate ions of SO_4^{2-} in the left side and in the right side of solution, respectively, while A.e^- , B.e^- indicate the electrons that are in the Zn and Cu electrode, respectively.

1. *Passage of Zn in the ionic form Zn^{2+} from the zinc electrode to the zinc sulphate solution A*, where $a = f(q_1, q_2, q)$ (according to Nerst's equation)

$$\frac{q_1\text{Zn}, q_2\text{Cu}, q\text{Zn}^{2+}}{(q_1 - a)\text{Zn}, q_2\text{Cu}, (q + a)\text{Zn}^{2+}, 2a\text{A.e}^-}.$$

2. *Passage of electrons in the wire (from Zn to Cu)*

$$\frac{(q + a)\text{A.e}^-, q\text{B.e}^-}{(q + a)\text{B.e}^-, q\text{A.e}^-}.$$

3. *Passage of Cu^{2+} ions, each with a couple of electrons, in metal form, from the copper sulphate to the copper electrode Cu*

$$\frac{q_1\text{Cu}, q_2\text{Cu}^{2+}, q\text{A.e}^-, (2a+q)\text{B.e}^-}{(q_1+a)\text{Cu}, (q_2-a)\text{Cu}^{2+}, q\text{A.e}^-, q\text{B.e}^-}.$$

4. *Passage of SO_4^{2-} ions from the solution B to the solution A*

$$\frac{q\text{A.SO}_4^{2-}, (q+2a)\text{B.SO}_4^{2-}}{(q+a)\text{A.SO}_4^{2-}, (q+a)\text{B.SO}_4^{2-}}.$$

In the paper [26] metabolism was logically represented by means of a model extending the model *SEQ*. Here we suggest another natural tool to describe metabolic rules.

Consider a combinatorial rule r over an alphabet A such that r applies to the strings β_1, \dots, β_m and produces the strings $\gamma_1, \dots, \gamma_n$. Given a finite set M of strings of A^* such that $\beta_1, \dots, \beta_m \in M$ we say that the *metabolite* M transforms to the metabolite M' , according to the rule r , if

$$M' = (M \setminus \{\beta_1, \dots, \beta_m\}) \cup \{\gamma_1, \dots, \gamma_n\}$$

where \setminus denotes the difference between sets.

Given a set R of rules over an alphabet A and a set B of strings of A^* , the set

$$\Gamma(B, R)$$

of *metabolic chains* of *axioms* B and rules R consists of finite sequences μ such that $\mu(1) = B$ and, if n is the length of μ , for every $1 < i \leq n$, $\mu(i-1)$ is a metabolite which transforms to $\mu(i)$ according to some rule $r \in R$. We write $\alpha \in \mu$ if $\alpha \in \mu(i)$ for some $1 \leq i \leq n$.

A *metabolic system* can be defined following the same pattern as in the definition of a derivation system.

Definition 7.1. A metabolic system \mathcal{M} is given by

$$\mathcal{M} = (A, T, B, R, C)$$

where A is a finite alphabet and T a subset of A , of terminal symbols; B is a finite set of initial strings over the alphabet A (the initial metabolite); R is a finite set of combinatorial rules over the alphabet A , and C is a subset of $\Gamma(B, R)$ of well-formed metabolic chains. The language $L(\mathcal{M})$ derived by \mathcal{M} is given by

$$L(\mathcal{M}) = \{\alpha \in T^* \mid \alpha \in \mu, \quad \mu \in C\}.$$

In metabolic systems, a combinatorial rule with many conclusions cannot be generally simulated by many combinatorial rules with only one conclusion (although, as it was already remarked, this is true for derivation systems). A derivation system can be considered as a particular metabolic system which is *cumulative*, that is, in its rules the premises are all included in the conclusions.

It would be a simple exercise to translate the metabolic rules of a Daniell cell into rules of a suitable metabolic system.

A metabolic system $\mathcal{M} = (A, T, B, R, C)$ is said to be:

- *pure* if $T = A$;
- *extended* if $T \subset A$;
- *regulated* if $C \subset \Gamma(B, R)$;
- *unregulated* if $C = \Gamma(B, R)$.

Many other concepts could be defined which are more specifically related to the notion of metabolism: *termination*, *periodicity*, *stability* (with respect to a class of initial metabolites), *dependency* (between rules and/or metabolic chains), *reachability* (of some metabolites from others), *minimality* (of some metabolic chains reaching some terminal metabolites). We do not develop here these aspects, rather we mention a result of monoidal representability that follows the same pattern as logical translation of unregulated derivation systems.

Theorem 7.1 (Logical translation of unregulated metabolic systems). *For every unregulated metabolic system \mathcal{M} there exists a monoidal theory $TH_{\mathcal{M}}$ and a formula φ in its signature such that $(TH_{\mathcal{M}}, \varphi)$ monoidally represents $L(\mathcal{M})$.*

Proof. Introduce a symbol $\#$ that does not belong to A , and number the rules of the system. If the rule

$$\frac{p_1, p_2, \dots, p_m}{q_1, q_2, \dots, q_n}$$

has the number j encode it by the formulae

$$Der(x, p_1) \wedge \dots \wedge Der(x, p_m) \rightarrow Der(x\#, q_1) \wedge \dots \wedge Der(x\#, q_n) \wedge Use(x, j)$$

$$Use(x, j) \wedge Der(x, y) \wedge \neg(y = p_1) \wedge \neg(y = p_2) \wedge \dots \wedge \neg(y = p_m) \rightarrow Der(x\#, y).$$

When all the rules of the system are so encoded, consider the formula $Der(\#, \alpha)$ for every $\alpha \in B$; moreover, add the following formulae:

- $Term(x) \wedge Term(y) \rightarrow Term(xy)$
- $Term(x) \wedge Der(y, x) \rightarrow Lang(x)$.

The set of formulae so obtained constitute $TH_{\mathcal{M}}$, in fact we have:

$$\alpha \in L(\mathcal{M}) \Leftrightarrow TH_{\mathcal{M}} \models Lang(\alpha)$$

this means that the pair $(TH_{\mathcal{M}}, Lang)$ represents $L(\mathcal{M})$. The predicate Der is binary, and its first argument is a sort of temporal parameter for indicating the different stages in the evolution of the system (identify the instant i with the string constituted by i occurrences of $\#$). The premises of a rule are not alive to the next step, but are replaced by the conclusions; all the other elements not affected by the rule survive in the next step. \square

In an analogous way, we could extend to the unregulated metabolic systems the systematic logical translation theorem, already given for unregulated derivation systems. But, what is more important for developing symbolic representations of real phenomena, we could extend to metabolic systems some logical regulation mechanism, analogous to those already considered for derivation systems. In this manner, the application of combinatorial rules in metabolic chains could be regulated by means of formulae within monoidal theories.

8. Conclusions

A biological system is constituted, at any level, by a physico-chemical support and by an information system that regulates the life processes, according to different strategies (phylogenetic, ontogenetic, epigenetic), for general finalities (fitness, reproduction, evolution), and for specific functionalities of the organisms.

The discovery of the discrete nature of information in basilar life processes (DNA, protein synthesis, metabolism) gives to theoretical computer science a privileged status in the search for mathematical models which explain principles, peculiarities and causes of dysfunctions of these processes. The theory of formal languages seems the most natural field where these models can be found.

This paper shows that formal logic can provide tools for comparing different symbolic formalisms, for discovering their common structure, beyond any particular syntax, and for describing the essence of symbolic processes in info, in vitro, or in vivo.

References

- [1] L. Adleman, Molecular computation of solutions to combinatorial problems, *Science* 266 (1994) 1021–1024.
- [2] D. Beaver, Computing with DNA, *J. Comput. Biol.* (2:1) (1995) 1–7.
- [3] J.L. Bell, M. Machover, *A Course in Mathematical Logic*, North-Holland, Amsterdam, 1977.
- [4] G. Berry, G. Boudol, The chemical abstract machine, *Theoret. Comput. Sci.* 96 (1992) 217–248.
- [5] N. Chomsky, Three models for the description of language, *IRE Trans. Inform. Theory* IT-2 (1956) 113–124.
- [6] E. Csuhaj-Varju, J. Dassow, J. Kelemen, G. Păun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach Sciences Publishers, Singapore, 1994.
- [7] E. Csuhaj-Varju, L. Kari, G. Păun, Test tube distributed systems based on splicing, *Comput. AI* 15 (2–3) (1996) 211–232.
- [8] H.B. Curry, Calculuses and formal systems, *Dialectica* 12 (3/4) (1958) 249–272.
- [9] J. Dassow, V. Mitrana, On some operations suggested by genome evolution, in: R.B. Altman, A.K. Dunker, L. Hunter, T.E. Klein (Eds.), *Proc. 2nd Pacific Symp. on Biocomputing*, World Scientific, Singapore, 1997, pp. 97–108.
- [10] J. Dassow, V. Mitrana, A. Salomaa, Context-free evolutionary grammars and the structural language of nucleic acids, *Biosystems* 43 (1997) 169–177.
- [11] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory*, Springer, Berlin, 1989.
- [12] R.L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Addison-Wesley, Redwood City, 1989.
- [13] R. Freund, F. Wachtler, Universal systems with operations related to splicing, *Comput. AI* 15 (4) (1996) 273–293.

- [14] B.L. Hao, *Elementary Symbolic Dynamics and Chaos in Dissipative systems*, World Scientific, Singapore, 1989.
- [15] T. Head, Formal language theory and DNA: an analysis of the generative capacity of recombinant behaviors, *Bull. Math. Biol.* 49 (1987) 737–759.
- [16] T. Head, Splicing schemes and DNA, in: G. Rozenberg, A. Salomaa (Eds.), *Lindenmayer Systems: – Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, Springer, Berlin, 1992., pp. 371–383.
- [17] T. Head, G. Păun, D. Pixton, Language theory and molecular genetics, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Language Theory*, Vol. 3, Springer, Berlin, 1997, Vol. 2, 1997 (Chapter 7).
- [18] L. Kari, DNA computers: tomorrow's reality, *Bull. European Assoc. Theoret. Comput. Sci.*, 59 (1996) 256–266.
- [19] S.C. Kleene, Representation of events in nerve nets and finite automata in: *Automata Studies*, C.E. Shannon and J. McCarthy (Eds.), Princeton Univ. Press, Princeton, NJ, 1956, pp. 2–42.
- [20] S. Kobayashi, Y. Sakakibara, Multiple splicing systems and the universal computability, M.F.C.S. 98 Satellite Workshop on: Molecular Computing, Brno, August 25–26, 1998.
- [21] A.L. Lehninger, *Principles of Biochemistry*, Worth, New York, NY, 1982.
- [22] A. Lindenmayer, Mathematical models for cellular interaction in development, I and II, 18 (1968) 280–315.
- [23] R. Lipton, DNA solution of hard computational problems, *Science* 268 (1995) 542–545.
- [24] P. Lorenzen, *Metamathematik*, Bibliographisches Institut, Mannheim, 1962.
- [25] V. Manca, String rewriting and metabolism: a logical perspective, in: G. Păun (Ed.), *Computing with Bio-molecules, Theory and Experiments*, Springer, Singapore, 1998.
- [26] V. Manca, D. Martino, From string rewriting to logical metabolic systems, in: A. Salomaa, G. Păun (Eds.), *Grammatical Models of Multi-agent Systems*, Gordon and Breach Science Publishers, London, 1998.
- [27] S. Marcus, Contextual Grammars, *Revue. Roum. Math. Pures Appl.* (1969) 1525–1534.
- [28] S. Marcus, Language between computation and biology, in: G. Păun (Ed.), *Computing with Bio-molecules. Theory and Experiments*, Springer, Singapore, 1998, pp. 1–35.
- [29] G. Păun (ed), *Artificial Life*, Black Sea University Press, Bucharest, 1995.
- [30] G. Păun, *Marcus Contextual Grammars*, Kluwer Academic Publishers, Boston, Dordrecht, 1997.
- [31] G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing, New Computing Paradigms*, Springer, Berlin, 1998.
- [32] E.L. Post, Formal reduction of the general combinatorial decision problem, *Amer. J. Math.* 65 (1943) 197–215.
- [33] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, NY, 1980.
- [34] G. Rozenberg, A. Salomaa (Eds.), *Lindenmayer Systems: – Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, Springer, Berlin, 1992.
- [35] G. Rozenberg, A. Salomaa (Eds.), *Handbook of Language Theory*, Vol. 3, Springer, Berlin, 1997.
- [36] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.
- [37] C. Smoryński, *Logical Number Theory*, Springer, Berlin, 1991.
- [38] R.M. Smullyan, *Theory of Formal Systems*, Princeton Univ. Press, Princeton, NJ, 1961.
- [39] G. Sundholm, Systems of deduction, in: D. Gabbay, F. Guenther (Eds.), *Handbook of Philosophical Logic*, Vol. 1, 1983, pp. 133–188.
- [40] A.M. Turing, The chemical basis of morphogenesis, *Philos. Trans. Roy. Soc.* 237 (1952) 37–72, also in *Collected Works of A.M. Turing*, in: P.T. Saunders (Ed.), *Morphogenesis*, North-Holland, Amsterdam, 1992, pp. 1–37.